John von Neumann Institute for Computing

Tracing the Cache Behavior of Data Structures in Fortran Applications

L. Barabas, R. Müller-Pfefferkorn, W.E. Nagel,
R. Neumann

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 211-218, 2006.

http://www.fz-juelich.de/nic-series/volume33

Tracing the Cache Behavior of Data Structures in Fortran Applications

Laszlo Barabas[a], Ralph Müller-Pfefferkorn[a], Wolfgang E. Nagel[a], Reinhard Neumann[a]

[a]Center for Information Services and High Performance Computing (ZIH)
Dresden University of Technology
D-01062 Dresden, Germany
{barabas,mueller-pfefferkorn,nagel,neumann}@zhr.tu-dresden.de

In an application, data access can become a major performance bottleneck if the memory hierarchy of the underlying hardware architecture is not taken into account. The only way to gain deeper insight of an applications memory usage is to measure its data access behavior with hardware counters. From the programmer's point of view such performance data (like cache misses or hits) have to be linked to the data structure causing it. The name of a data structure is the only point of reference the user has and the only point where he can apply optimizations.

In the project EP-Cache tools for Fortran applications were developed to monitor and to link hardware counter information with data structures. This includes an appropriate visualization of the gathered information. Parallelism using the OpenMP programming paradigm is also supported.

1. Introduction

"Processor performance is not an issue." At least in the last years and, probably, up to 2012 this statement was and is still valid. Following Moore's law the speed and capabilities of modern microprocessors have increased significantly in the past and will do so in the near future.

Nevertheless, the overall performance of a computer depends on more than pure CPU power. Memory access is (still) a bottleneck. Memory performance increases only by about 7% per year. The concept of hierarchical cache levels providing faster access to parts of data improves the situation but needs a strict adoption of programming to its paradigms. Compilers and other optimization tools might support programmers to reach this goal. But this job is very complex and there are many influencing factors like data layout, details of the hardware structure etc. Only in a multistage development process of "coding - testing - monitoring" can an optimum be achieved.

In applications for numerical calculations or for data processing large data arrays, vectors or other structures are usually used. The employment of data access techniques which exactly reflect the memory hierarchy of caches and main memory of the underlying hardware system are crucial to gain the best performance in such cases. The "only" problem is to know, how a code makes use of the memory hierarchy. This is a nontrivial problem, as in most cases software is too complex for theoretical consideration to reveal the true nature of the data access at the instruction level. The only way to find that out is to measure the system's data access behavior.

Hardware counters that measure cache and memory hits and misses are a well known mechanism to gather data access information. Many tools were written to collect these information

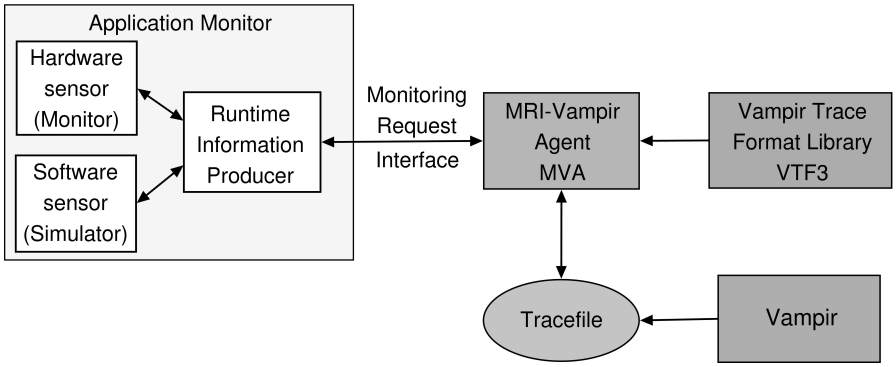

Figure 1. Architecture of the hardware and simulation based monitoring system

as profiling data, giving a summary about an applications behavior. More sophisticated tools even provide counter data for regions of a code allowing a more structural view on its data accesses. But to draw a direct line between a single data structure in a code region and a counter information is a problem only few profiling tools can solve [5].

For years, performance monitoring of programs was successfully done by writing out detailed information to trace files during program execution. In a second step, and (usually) after the execution had finshed, the collected data were analyzed using visualization techniques. Due to the complexity of modern applications the visualization has become an essential part of a performance analysis and often the only way to understand in detail what was going on during the program run.

The aim of the project EP-Cache[1] [2] was to develop tools to support Fortran programmers in the monitoring and optimization steps, specializing on data structures and their cache access.

In this paper, we want to present some of the results of these efforts, focusing on the tools for monitoring data structure related cache performance and their visualization. In the sections 2 and 3 we want to describe the two approaches to collect data structure related performance data. Section 4 reports about the extensions needed for the tracing infrastructure that was used to record the performance data. Finally, section 5 describes the new visualizations that were implemented into *Vampir* [3], the well-known performance analysis suite, to effectively analyse the collected data.

2. The Hardware Monitor EPCMON

At the TU München, a hardware monitoring environment was developed to expand the classical hardware counter concept and to allow the measurement of data structure related information at runtime.

The hardware monitor can run in two modes. The *static mode* allows to count address specific events. Such events can be e.g. L1 misses for array A in one or several specific code regions. The *dynamic mode* enables a monitoring of accesses to address ranges. It delivers the data in the form of a histogram with a configurable granularity as multiples of cache lines. An example could be a histogram with 30 bins gathering L1 cache hits for an integer array C[120] in a specific code region. With a cache line size of 8 bytes and an integer length of 2

[1]funded by the German Federal Ministry of Education and Research (BMBF) under contract 01IRB04

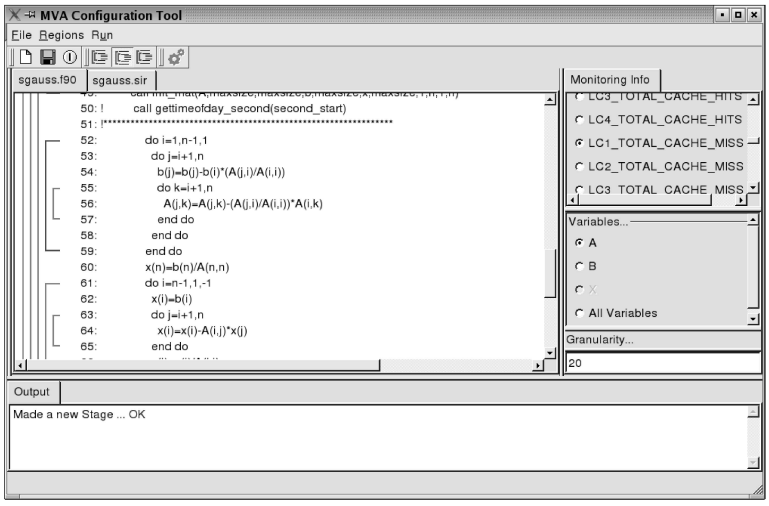

Figure 2. The graphical user interface MVAconfigure eases the specification of monitoring requests

bytes one bin would hold the values for 4 elements of array C.

As the hardware monitor is still a concept currently, a simulation of the monitor was implemented to test and verify its concepts. It is based on the Valgrind [8] runtime instrumentation framework for IA-32 architectures and allows a on-the fly cache simulation for OpenMP programs.

To define regions in the source code for the measurement of the performance data and to map the virtual addresses or address ranges to data structures instrumentation of the source code is used.

Both, the hardware monitor and the simulation can be accessed via a common interface - the *Monitoring Request Interface* (MRI) [4]. MRI allows a consumer to specify monitoring requests and to retrieve the monitor data. A request can look like: Monitor the L2 cache misses for array A in the file matmul.f90 in the region from lines 10 to 14 and write the data in a histogram with 20 bins!

A more detailed description of the hardware monitor and the simulator can be found in [6].

To steer the monitor and to write out the performace data to tracefiles (see section 4) a tool was implemented (MRI-Vampir-Agent, MVA). MVA reads in a configuration file with monitoring requests, transfers the requests to the monitor, reads out the performance information and writes them to a tracefile. Beside the new data structure related counter information (see section 4) "traditional" trace events (like enter and exit of regions or OpenMP related events) are written out. Fig. 1 illustrates the basic architectural concept of the monitor in combination with the *VTF3* tracing infrastructure and a *Vampir* based performance analysis (see sections 4 and 5).

To create the configuration file with the monitor requests a graphical user interface (MVAconfigure) was written. The user simply loads his source code into MVAconfigure and with some clicks he can select the regions, the data structures, the hardware counter and other configurations for the requests (see fig. 2).

3. Source Code Instrumentation Based Monitoring

Another approach to gather data structure related performance data does not use special hardware or simulation. It is based on the ADAPTOR compilation system ([1], SCAI FhG Sankt Augustin) and combines three techniques to get the needed information.

- source code instrumentation
- precise event-based sampling (PEBS) of hardware counters
- mapping of addresses to data structures

The original source code is instrumented automatically for tracing using ADAPTOR, compiled and run. The user can select the regions, counters and data structures for monitoring.

In a region, where the data access behavior should be observed, a precise event-based sampling is started. The technique is, that e.g for every 200th L1 cache miss an interrupt is generated. Modern processors (as the Pentium IV or the Itanium) then allow the readout of the (precise) current instruction pointer and all general purpose registers. With this information, one can identify the current instruction and compute the data addresses used in it. Such, a data address profile for specific events (like cache misses) in one or more specific regions can be collected.

The final step is to translate the data addresses to the corresponding data structure in the source code. This is done with ADAPTOR's runtime memory management. By instrumenting the source code the addresses of the data structures can be determined at runtime and a reverse mapping can be applied. Thus, dynamically allocated memory can also be included.

Currently, the monitoring runs on two platforms that support precise event-based sampling: Pentium IV and Xeon machines (Linux with the perfctr [9] and hardmeter [11] packages) and on Itanium machines (Linux with libpfm [7]).

4. The VTF3 trace infrastructure

For both the hardware monitor/simulator and the instrumentation based monitoring, the trace data is stored in a *VTF3* tracefile [10]. *VTF3* is a trace infrastructure developed at ZIH. *VTF3* can be read and visualized by *Vampir* for post-mortem performance analysis.

VTF3 provides definitions and records for various kinds of trace events, such as the definition of the computing environment (e.g. number of CPUs or CPU grouping), enter/exit of code regions, hardware counter data, MPI communication, OpenMP events etc. They are stored together with a time stamp to get a temporal view of the applications run.

In the EP-Cache project *VTF3* was extended to store records for data structure related performance data. 4 new records were defined and implemented. They allow the definition of data structures and histograms.

For data structures, the name and source code locations (file name, line numbers) can be recorded.

A histogram represents a vector of (counter) values. For each vector element (each bin of the histogram) the data structures, which the counter values were collected for, are stored. Other general information for the histogram includes timestamps (start and end time of the data collection), the kind of counter information, process information and more.

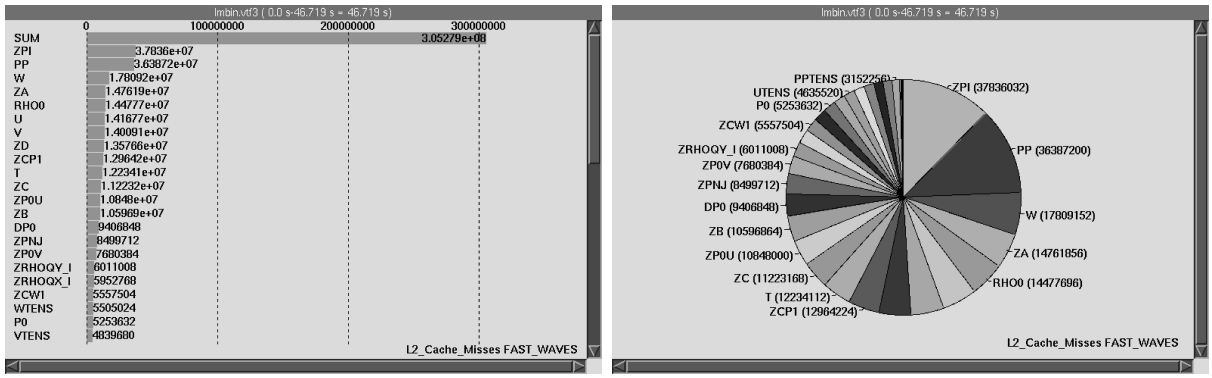

Figure 3. Global summary of the cache usage of the monitored data structures, bar and pie representation

5. Visualizing data structure related performance data

Vampir is a visualization and performance analysis environment, which has been developed at the ZIH for many years. *Vampir* is used at many HPC centers worldwide. It reads information from tracefiles and transfers them to a variety of graphical displays (e.g. timelines, state diagrams, statistics . . .), supporting users in the analysis and optimization of their programs.

To visualize the new data structure related performance data in a typical *Vampir*-like style, new displays were developed and implemented. Thus, various possibilities are available to the user to analyse performance losses resulting from the cache behavior of his application.

Depending on the users choice, the data are visualized either time dependent with as a so called "timeline" or as summaries. The new features are described in the following.

5.1. Summary data

A first glance on the overall cache performance of the data structures of a program gives the display *Cache Data Summary*. It summarizes the values of the performance counters for the monitored data structures. The information can be shown either in a bar or a pie representation (fig. 3). In the bar presentation the data structures can be sorted depending on the counter values. The pie representation gives the possibility to view the fraction each data structure contributes.

In the *Cache Bin Statistics* (fig. 4(a)) the counter values are displayed as a histogram, where each bin represents either one data structure, parts of a data structure or several data structures.

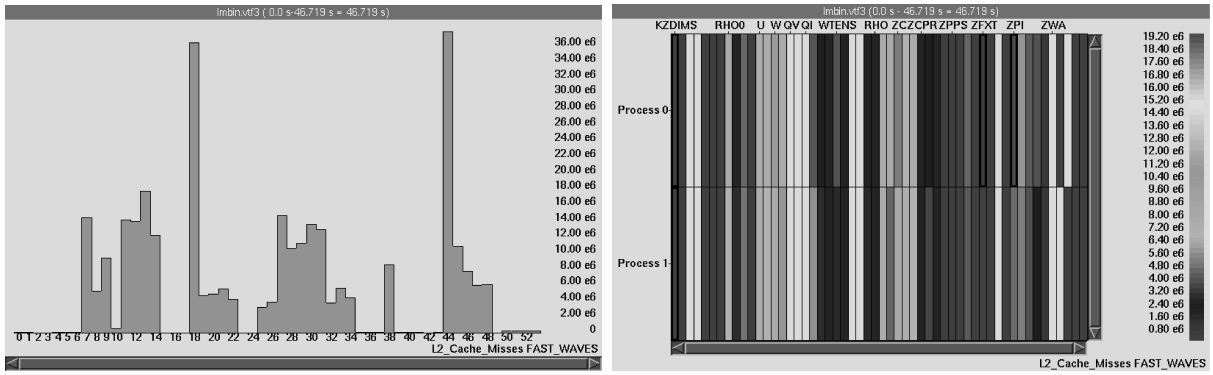

Figure 4. Cache Bin Statistics (left) and Cache Data Statistics for an example with 2 processes (right)

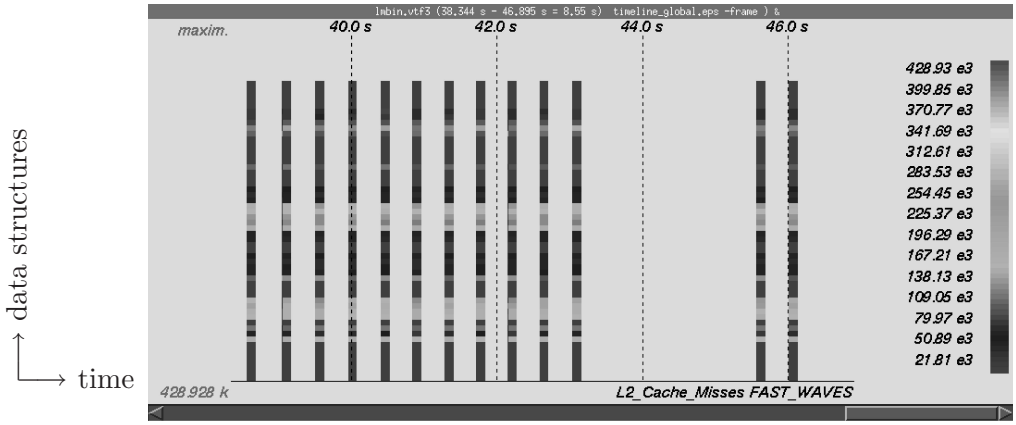

Figure 5. Global cache timeline (summarized over all processes)

The distribution of counter values to processes and data structures is shown in the *Cache Data Statistics* display. The counter values are colorcoded for each bin. Thus, a direct comparison of the cache behavior of the data structures in the individual processes is possible.

For all of the above displays the shown summaries depend on the timeframe the user selected in any timeline - zooming in or out in time, recalculates and adjusts the displays. Additionaly, individual processes can be filtered out for the calculation of the statistics.

5.2. Cache Timelines

The most interesting new feature are the Cache Timelines that visualize the temporal development of the cache behavior of the monitored data structures. The Cache Timelines show the histograms of counter values of the data structures (ordinate) as a function of time (abscissa). The time spread of the histograms is given by the start and end time of the monitoring in a region. The counter values of the individual bins are colorcoded.

In the *Colorcoded Cache Timeline* the counter values, summarized over all processes, are displayed. In the example in figure 5, which shows the L2 misses of all data structures of one subroutine (FAST_WAVES), one can clearly see, that there are some data structures with a large number of L2 cache misses and others with almost no misses. But over time, the cache behavior of the single data structures does not change significantly.

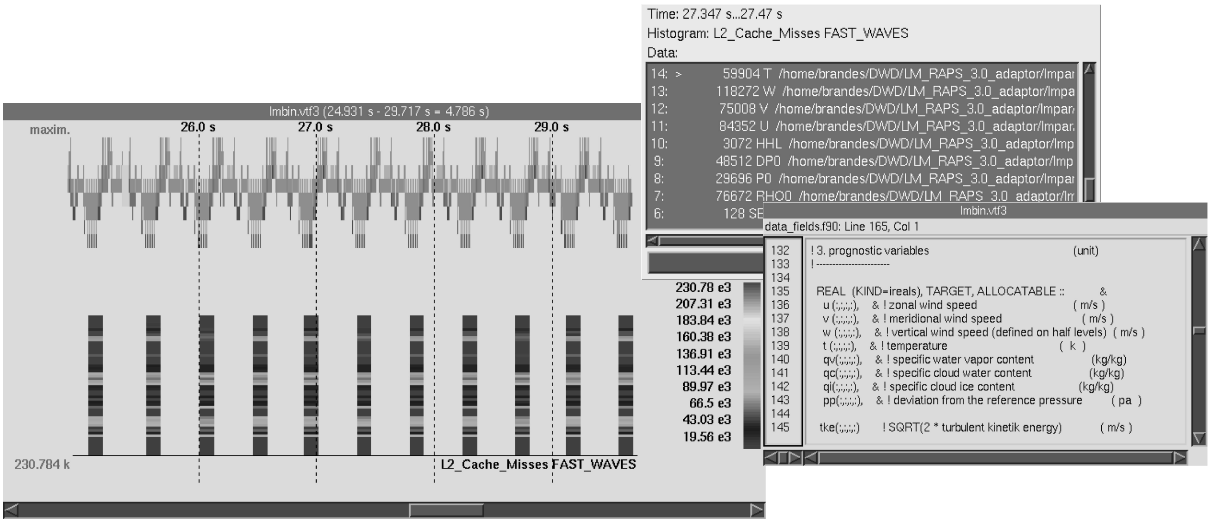

Figure 6. Process cache timeline with histogram information window and source view

If the user is interested in the data of a single process only, he can select one (in the *Vampir* timeline display) and take a look at the *Process Cache Timeline* (fig. 6). Below the call stack of the selected process the Cache Timeline is displayed. The combination of both allows the user to better analyse and understand the correlation between the call of single code regions and the cache behavior of data structures in these regions.

5.3. Connection to source code

In all of the displays described above the user has direct access to information about the data structures. Clicking e.g. on a bin in a histogram opens up a window with the name of the data structure monitored and the corresponding counter value (see fig. 6). A click on the name of the data structure brings up another window containing the relevant source code.

6. An Illustrating Example

The more complex a code is the more difficult it is to specify exactly the reason for a performance loss. Especially in numerical calculations or in simulations where complex calculations are done (often on one code line), the kind of access to a single data structure can cause a significant performance decrease.

In the following (simple, and thus comprehensible) example we want to illustrate the benefits the monitoring of data structure related performance data provides.

```
!ADDITION OF MATRIX ELEMENTS
!REGION 1
DO I=1,N
   DO J=1,N
     S=S+A(I,J)+B(J,I)
   ENDDO
ENDDO
!REGION 2
DO J=1,N
   DO I=1,N
     S=S+A(I,J)+B(I,J)
   ENDDO
ENDDO
```

In region 1, array A is accessed in the wrong order - Fortran stores arrays columnwise in memory. Array B is accessed in the correct order. In region 2 both A and B are accessed correctly in columnwise order. Monitoring the L1 cache misses without a correlation to the data structures, reveals a cache access problem in region 1 (fig. 7a) - but no sign of what is causing it. (There are still misses in region 2, because A and B do not fit into the cache.) Fig. 7b shows the colorcoded cache timeline of the L1 cache misses for array A and B separately. It clearly indicates, that the number of L1 cache misses for A in region 1 (wrong order) is much larger than the one in region 2. For array B the number of misses is the same for both regions. Thus, it shows clearly that only the access to A is the problem.

7. Summary

In the EP-Cache project a number of tools were developed that allow the monitoring of data structure related cache performance counters in Fortran applications. New extensions enable the *VTF3* trace infrastructure to record such information, e.g as histograms, for a post-mortem analysis. The performance-analysis environment *Vampir* was equipped with displays

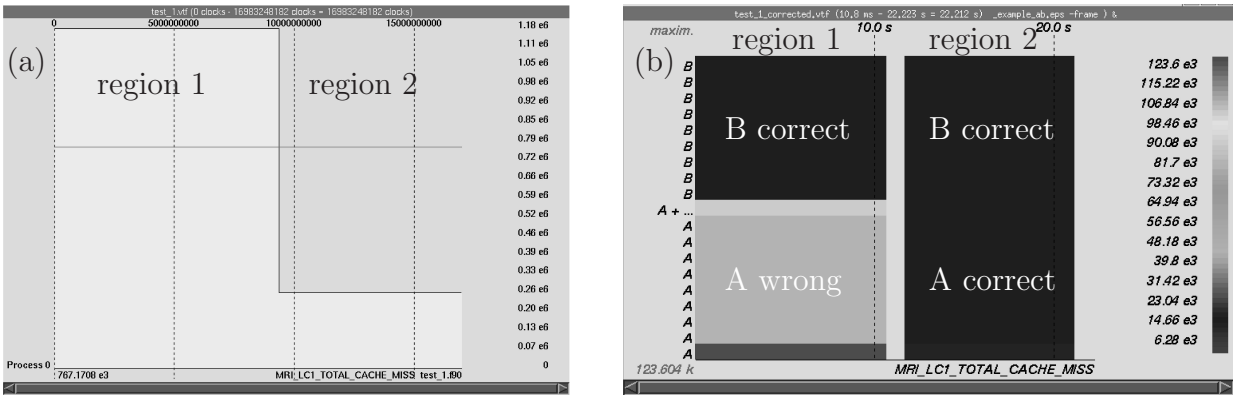

Figure 7. (a) counter timeline: L1 cache misses without correlation to data structures; (b) colorcoded cache timeline for L1 cache misses: array A - wrong access order in first, correct access order in second region; correct access order for array B in both regions

to visualize the new data structure related cache performance data. This enables users to analyse their code regarding the cache behavior of individual data structures, thus providing the posssibility to optimize the data access patterns at source code level.

References

[1] T. Brandes. *ADAPTOR - Automatic DAta Parallelism TranslaTOR.* Institute for Algorithms and Scientific Computing (SCAI FhG), Sankt Augustin, 2000. http://www.scai.fhg.de/index.php?id=291&L=1.

[2] Th. Brandes, H. Schwamborn, M. Gerndt, J. Jeitner, E. Kereku, W. Karl, M. Schulz, J. Tao, H. Brunst, W. E. Nagel, R. Neumann, R. Müller-Pfefferkorn, B. Trenkler, and H.-Ch. Hoppe. Monitoring Cache Behavior on Parallel SMP Architectures and Related Programming Tools. *Journal on Future Generation Computing Systems*, 2005.

[3] Holger Brunst, Wolfgang E. Nagel, and Allen D. Malony. A distributed performance analysis architecture for clusters. In *IEEE International Conference on Cluster Computing, Cluster 2003*, pages 73–81, Hong Kong, China, December 2003. IEEE Computer Society.

[4] M. Gerndt and E. Kereku. Monitoring request interface version 1.0. Technical report, TU München, 2004. http://wwwbode.in.tum.de/∼ kereku/epcache/pub/MRI.pdf.

[5] M. Itzkowitz, B. J. N. Wylie, Ch. Aoki, and N. Kosche. Memory Profiling using Hardware Counters. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, Washington DC, USA, 2003. IEEE Computer Society.

[6] E. Kereku, T. Li, M. Gerndt, and J. Weidendorfer. A Selective Data Structure Monitoring Environment for Fortran OpenMP Programs. In D. Laforenza M. Danelutto, M. Vanneschi, editor, *Euro-Par 2004 Parallel Processing*, page 133, Pisa, Italy, 2004. Springer-Verlag.

[7] HP Labs. *Performance Analysis Tools for the IA-64.* http://www.hpl.hp.com/research/linux/perfmon/.

[8] N. Nethercote and J. Seward. Valgrind: A program supervision framework. In *Proceedings of the Third Workshop on Runtime Verification (RV03)*, Boulder, Colorado, USA, July 2003.

[9] M. Pettersson. *Linux x86 Performance-Monitoring Counters Driver.* http://www.csd.uu.se/ mikpe/linux/perfctr/.

[10] S. Seidl, A. Knüpfer, and R. Müller-Pfefferkorn. VTF3 - A Fast Vampir Trace File Low-Level Management Library. Technical report, ZIH TU Dresden, 2004.

[11] K. Takehiro and Hiro Yoshioka. *Hardmeter - a memory profiling tool*, 2003. http://sourceforge.jp/projects/hardmeter.